IS12 - Introduction to Programming

Lecture 7: Introduction to C

Peter Brusilovsky
http://www2.sis.pitt.edu/~peterb/0012-051/

Why C?

- Modular procedural language with arrays, structures, and references
- C vs. Pascal
  - modern, portable, better textbooks and tools
  - employment prospects (C, C++, Java)
- C vs. C++ or Java
  - small, clean, simple
  - can support learning data structures without learning too much language extras (IS15)
  - provides an easy transfer to C++ and Java

Books for C

- Perry: Absolute Beginner's Guide to C
- Other
  - C for Dummies
  - Kernighan and Ritchie
  - Deitel and Deitel
- What's about price?
  - Use library
  - Multiple free tutorials on the Web
  - Knowledge Sea system

Learning C

- Be careful, read your programs!
  - The basic philosophy of C: "programmers know what they are doing" - K&R2, p.3
- Ask questions in CourseWeb forums
- Meet TA and your instructor during the office hours
- Practice, practice, practice!
  - Run all examples, modify it
  - Solve problems, check yourself on quizzes

Commands and data

- Components of a program
  - Objects (data)
  - Commands (instructions)
- This is true on several levels
- Basic features of a machine language or a programming language:
  - Ways to represent objects - data types
  - Ways to act on information - operations

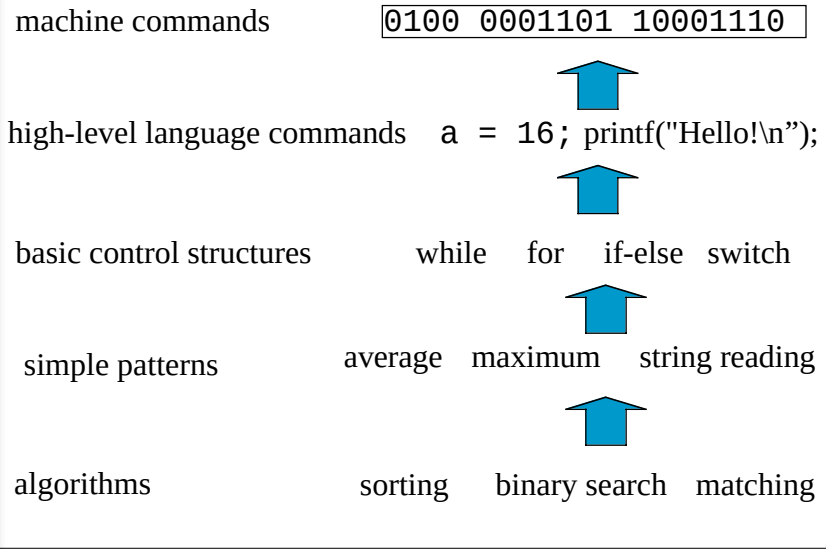
Karel vs. C

- Data
  - Karel - city, beepers, walls
  - C - numbers and symbols in memory
- Commands
  - Karel - move, turnleft
  - C - add, print…
- Karel operates in a visible word outside; C programs work invisibly inside

Information Representation

- Computer store information digitally in binary format
- Ultimately everything is ones and zeroes
  - characters, numbers
  - instructions (programs!!)
  - pictures, video
- Binary arithmetic E.g., 7 = 00000111
  99 = 01100011

From Commands to Algorithms



C Program Syntax

- Most C programs have the following (at minimum):

```
main ([program arguments])
{
    <one or more statements>
}
```

- Every program must have a “main” function. Note that C is case sensitive (Main is not the same as MAIN or main).

Hello World Program

```
/* This is our first program */
#include <stdio.h>

void main()
{
     printf("Hello World!\n");
}
```

Dissection adapted From S. Pilachewski

Hello World - dissected

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- This is a *comment*. Comments are written not for computers but for humans. The computer will ignore everything between /* and */. Humans need comments to understand the program. (Why?)

Hello World - dissected

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- *#include* is a *command* which tells the compiler that the standard input / output library will be used. Thus printf will be recognized as a standard output function

Hello World - main() Function

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- Execution of a C program always begins at the ***main()*** function. Every C program must have one (only one) main() function.

Hello World - the Braces

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- The open brace (**{**) marks the beginning of the function body, which is one or more program statements which perform some task.
- The closing brace (**}**) marks the end of the function body.

Hello World - the *printf* statement

```
/* This is our first program */
#include <stdio.h>

main()
{
      printf("Hello World!\n");
}
```

- This *statement* is a function call to the *printf* function in the C Standard I/O Library. It displays the message which is the argument to the function. The **\n** denotes the newline. We can use printf because we told the compiler to use Standard I/O Library in *#include*

Hello World - the *semicolon*

```
/* This is our first program */
#include <stdio.h>

main()
{
      printf("Hello World!\n");
}
```

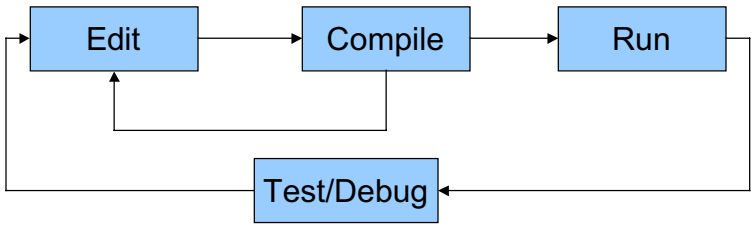
- The semicolon marks the *end* of a C program statement.

The edit-compile loop again

1. Edit program
2. Compile program
3. If there are errors, fix and go back to 1
   - you have got syntax error
   - fix and go back to 1
4. Run it
5. If it produce wrong results
   - you have got semantic error
   - find the source of the error (debug)
   - fix and go back to 1

The iterative nature of program design

The "programming in small" loop



Programming tools

- Editor
  - PC: Crimson Editor, PFE
  - Mac: BBEdit
  - Unix: emacs, vi, nedit
- Compiler
  - PC: LCC
  - Unix: cc/gcc
- IDE
  - LCC-Win, MS Visual C++, Borland, Leonardo

Tools for C

- Recommended
  - LCC-WIN and Microsoft Visual C++ IDE for Windows
  - Leonardo for Macintosh
- Other options
  - Other IDEs: Borland,…
  - Editor/Compiler pair for Win and Unix
    - Use some good programming editor

IDE for C vs. IDE for C++

- Users of Visual C++ and Borland, beware! C++ is not C!
- Almost every C++ environment has an option to set it working in strict C. Do it!
- Avoid typical errors - do not lose points!
  - Comments: /* */ vs. //
  - Variable declarations: At the beginning of a function or anywhere (even inside a loop)

Hello World - Syntax Error

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n")
}
```

- The semicolon is missing
- What happens when we compile this?

Hello World - Compile Error

- What happens when we compile this?

```
(5) unixs4 $ cc hello-nosemi.c
"hello-nosemi.c", line 6: syntax error
  before or at: }
(6) unixs4 $
```

Hello World - Experiment 2

```
/* This is our first program */
#include <stdio.h>

main()
{
     /* note the absence of \n */
     printf("Hello World");
}
```

- What happens when we run this?

Hello World - Experiment 2

- What happens when we run this?

```
(13) unixs4 $ cc hello-
  nonewline.c
(14) unixs4 $ ./a.out
Hello World(15) unixs4 $
```

Hello World - Experiment 3

```
/* This is our first program */
#include <stdio.h>

main()
{
      /* note the extra \n */
      printf("Hello\n World\n");
}
```

- What happens when we run this?

Hello World - Experiment 3

- What happens when we run this?

```
(17) unixs4 $ cc hello-
  extranewline.c
(18) unixs4 $ ./a.out
Hello
 World
(19) unixs4 $
```

Before next lecture:

- Install / try LCC-Win, Visual C++ or other editor/compiler or IDE
- Get and check the books
- Compile and run Hello World program
- Experiment! Print your name, make errors, correct them!
- Readings: Perry, Chapter 1; Chapter 2 (until “Kinds of Data”); Chapter 3 (first reading)